

**Department of Electrical & Computer Engineering**

**Second Semester (2022/2023)**

**ENCS3340, ARTIFICIAL INTELLIGENCE**

**Project # 1**

**Prepared by:**

**Hidaya Mustafa 1201910**

**Adam Nassan 1202076**

**Instructor: Dr. Ismail Khater**

**Dr. Yazan Abufarha**

**Date:20/6/2023**

**Section: 3,4**

# **Procedure:**

## **Project description**

This project implements a game called "Magnetic Cave" where two players take turns placing their symbols on an 8x8 board. The game can be played manually or with an AI opponent using the minimax algorithm. The code includes functions for checking winning conditions, evaluating game positions, and making intelligent moves for the AI player within a time limit of 3 seconds.

The game is played on an 8x8 board represented by a 2D array called Plac. Each cell of the board can be empty (0), occupied by player 1 (■), or occupied by player 2 (□).

The game menu is displayed, and the user is prompted to enter their choice (1, 2, 3, or 4) for different game modes.

Depending on the user's choice, the game can be played in different modes:

Mode 1: Manual entry for both players' moves. The players take turns entering the row and column of the position they want to place their symbol. The board is displayed after each move, and the game checks for a winner or a tie.

Mode 2: Manual entry for player 1 and automatic moves for player 2. Player 1 enters their move, and then an AI algorithm called "minimax" is used to determine the best move for player 2. The board is displayed after each move, and The game checks for a winner or a tie.

Mode 3: Manual entry for player 2 and automatic moves for player 1. Similar to mode 2, but player 2 enters their move, and the AI algorithm determines the best move for player 1.

Finally, Mode 4 allows the player to quit the game.

The game board is represented by a 2D array, and each cell can be empty (0), occupied by player 1 (■), or occupied by player 2 (□). Several functions are defined to manage the game logic. The displayBoard function prints the current state of the board. The initBord function initializes the board. The enterValue function places a player's symbol on the board at a specified position. The isValid function checks if a position is a valid move. The is\_game\_Over function determines if the game is over. The checkGameWinner function checks for a winner. The isBoardFull function checks if the board is full. The minimax function is the AI algorithm used to determine the best move for the computer player, and it uses a heuristic function to evaluate game positions.

Overall, the project allows players to play the "Magnetic Cave" game in different modes, including playing against an AI opponent using the minimax algorithm. The code efficiently manages the game logic, evaluates positions, and makes intelligent moves for the AI player within a specified time limit.

## **Heuristic description:**

The heuristic() function calculates a score for the current game state. It takes into account the current player (player) and the opponent player (opponent), it starts by checking if there is a winner in the game by calling the checkGameWinner() function, if player 1 wins (winner == 1), the score is set to the maximum possible value (Integer.MAX\_VALUE), if player 2 wins (winner == -1), the score is set to the minimum possible value (Integer.MIN\_VALUE), if there is no winner but the game board is full (indicating a draw), the score is set to 0, if none of the above conditions are met, it proceeds to evaluate the board based on the player's score (playerScore) and the opponent's score (opponentScore), the player's score and opponent's score are calculated by calling the evaluatePlayerScore() function for each player, finally, the score is determined by subtracting the opponent's score from the player's score (score = playerScore - opponentScore), which provides an evaluation of the current game state, it's important to note that the evaluatePlayerScore() function should be implemented to evaluate the score based on factors such as the number of consecutive pieces in each direction and any other relevant considerations specific to your game.

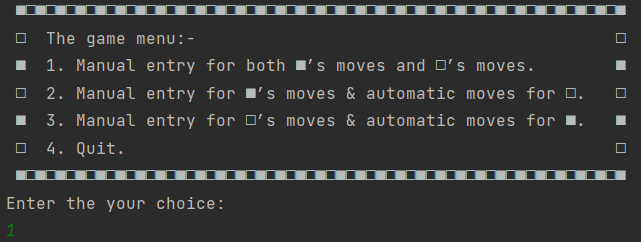
# **Result :**

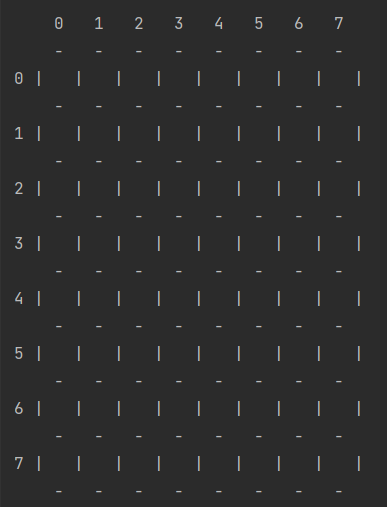
## **In Mode 1:**

where both players took turns manually entering their moves. The game board was displayed after each move, allowing the players to see the current state of the game.

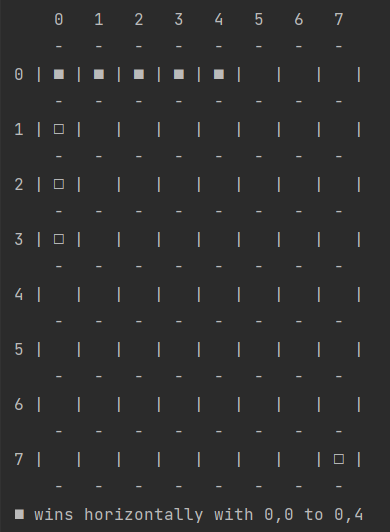
The moves were entered by specifying the row and column of the desired position on the board. The board was then updated accordingly, with Player 1 represented by the symbol "■" and Player 2 represented by the symbol "□".

After several moves, Player 1 (■) was able to create a horizontal line of symbols from position (0, 0) to (4, 0), resulting in a win. The game correctly recognized this winning condition and displayed the corresponding winning message.





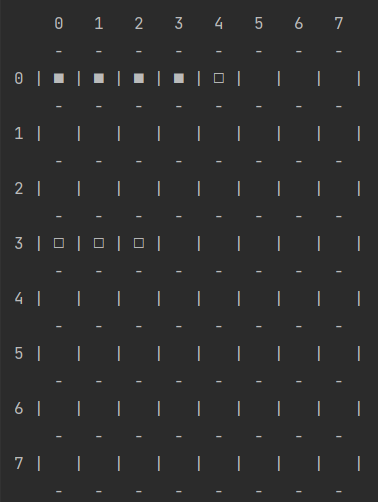
After executing the movements entered by the players:



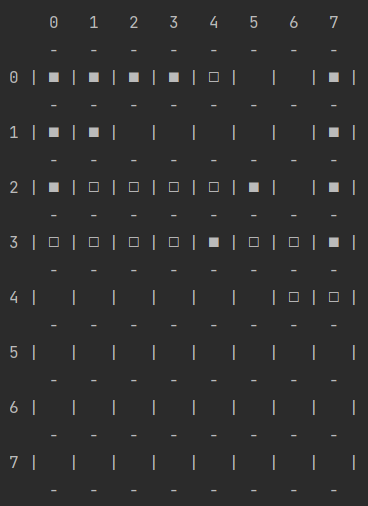
## **In Mode 2 :**

Here the first player (■) took the move manually and the second (□) took it automatically using our Ai functions (minimax and heuristics).

After some moves as you notice in the figure below the computer trying to win horizontally in the 3rd row, also when I tried to win using 1st row he rapidly prevented me.



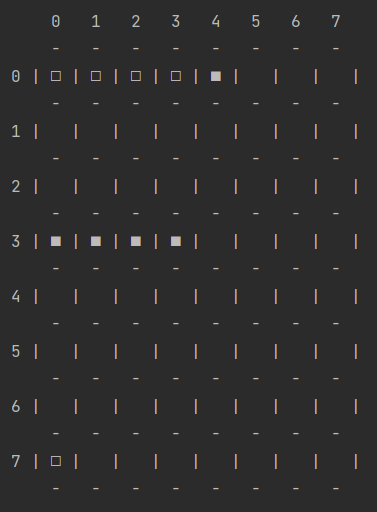
Then when continuing the game also the computer prevented me from winning vertically using the 7th column.

****

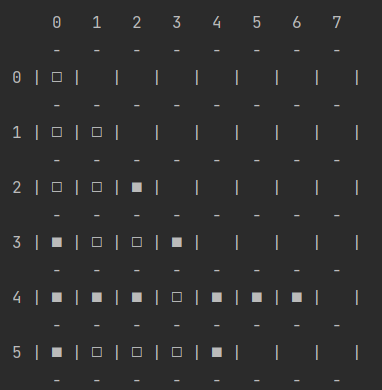
## **In Mode 3:**

Same as mode 2 but here the first player (■) took the move automatically and the second (□) took it manually.

In the figure shown below the computer also prevented me from winning horizontally using the 1st row.



Also in the figure below the computer prevented me from winning diagonally from 10 to 54.



Overall, the game progressed smoothly, and the players were able to interact with the game by entering their moves. The game mechanics, such as updating the board and checking for a winner, functioned as intended, providing an enjoyable and accurate gaming experience.